INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

L4 -2024 JANUARY BATCH

Module:4COSC006C.2 – SD I - Programming

Module Leader: Mr. Guhanathan Puravi

Type of Assignment: Individual Course Work Part B

Submission Date: 15/04/2024

Group ID: Group E

Student Name	Student_ID	Westminster ID
N.S. Atugoda	20232130	W2083577

❖ Acknowledgment

I would like to express my deep appreciation to the Informatics Institute of Technology for guiding me through this assessment and giving me invaluable experience. I am so grateful to our lecturers for their unwavering support and for providing us with the necessary knowledge and tools to make this report a success.

❖ List of figures

Table of Contents

1. Problem specification.

1.1. Overview

Building on your knowledge of Python, dictionaries, and file I/O, your next challenge is to enhance the Personal Finance Tracker by developing a graphical user interface (GUI) using Tkinter. This advanced version should not only display the information from a provided JSON file but also incorporate object-oriented programming (OOP) concepts for the GUI components. Additionally, your application will include a search function and a sorting feature, similar to file explorer, to manage and analyze financial transactions more effectively.

1.2. Objectives

1. Integrate a GUI using Tkinter and OOP concepts.
2. Load and display data from a JSON file upon GUI invocation.
3. Implement search and sorting functionalities within the GUI.
4. Ensure the application is user-friendly and robust.

2. Pseudocode.

FinanceTrackerCLI:

```
1. Initialization:
      • Initialize the filename for transactions.
      • Load transactions from json file.
2. Load transactions:
      • Check if the json file exists.
      • If it does:
            Load transactions from file.
      • Else:
            Initialize transactions as an empty dictionary.

3. Save transactions:
      • Save transactions to the json file.
4. Run CLI:
      • Display the main menu in the loop.
      • Based on user choice, execute different functions.
            - Show all transactions.
            - Add a new transaction.
            - Update a transaction.
            - Delete a transaction.
            - Display summary.
            - Launch GUI
            - Exit the program.
5. Show transactions:
      • Print all transactions groped by category.
6. Add transaction:
      • Prompt the user to enter category, date, and amount.
      • Add the new transaction to the transactions dictionary.
```

- Save transactions to the file.

7. Update transaction:
    - Prompt the user to enter the category and index of the transaction to update.
    - Update the transactions with the new amount and date.
    - Print “transaction updated successfully!”
8. Delete transaction:
    - Prompt the user to enter the category and index.
    - Delete the transactions from the transactions dictionary.
    - Print” transaction deleted.”
9. Display summary:
    - Calculate the total amount for each category.
    - Print the summary of total amounts of each category and overall total.
10. Launch GUI:
    - Initialize a Tkinter window and pass the transaction data.
    - Display the GUI.

FinanceTrackerGUI:

1. Initialization:
    - Initialize the Tkinter window and set its title.
    - Initialize transactions data and sorting variables.
    - Create widgets for the GUI.
2. Create widgets:
    - Create treeview widget to display transactions.
    - Add headings for category, amount, and date.
    - Add a vertical scrollbar for the treeview.
    - Create an entry field and a button for search transactions.
3. Load transactions:
    - Load transactions from the json file.
    - Display the transactions in the treeview.

4. Display transactions:
    - Remove existing from the treeview.
    - Add transactions to the treeview.
5. Search transactions:
    - Get the search criteria entered by the user.
    - Filter transactions based on the search criteria.
    - Display filtered transactions in the treeview.
6. Sort transactions:
    - Sort transactions based on the selected column.
    - Update the treeview with sorted transactions.
7. Save transactions:
    - Save transactions to the json file.
8. Main menu function:
    - Create an instance of FinanceTreckerCLI
    - Run the CLI application.

3. Prepared code.

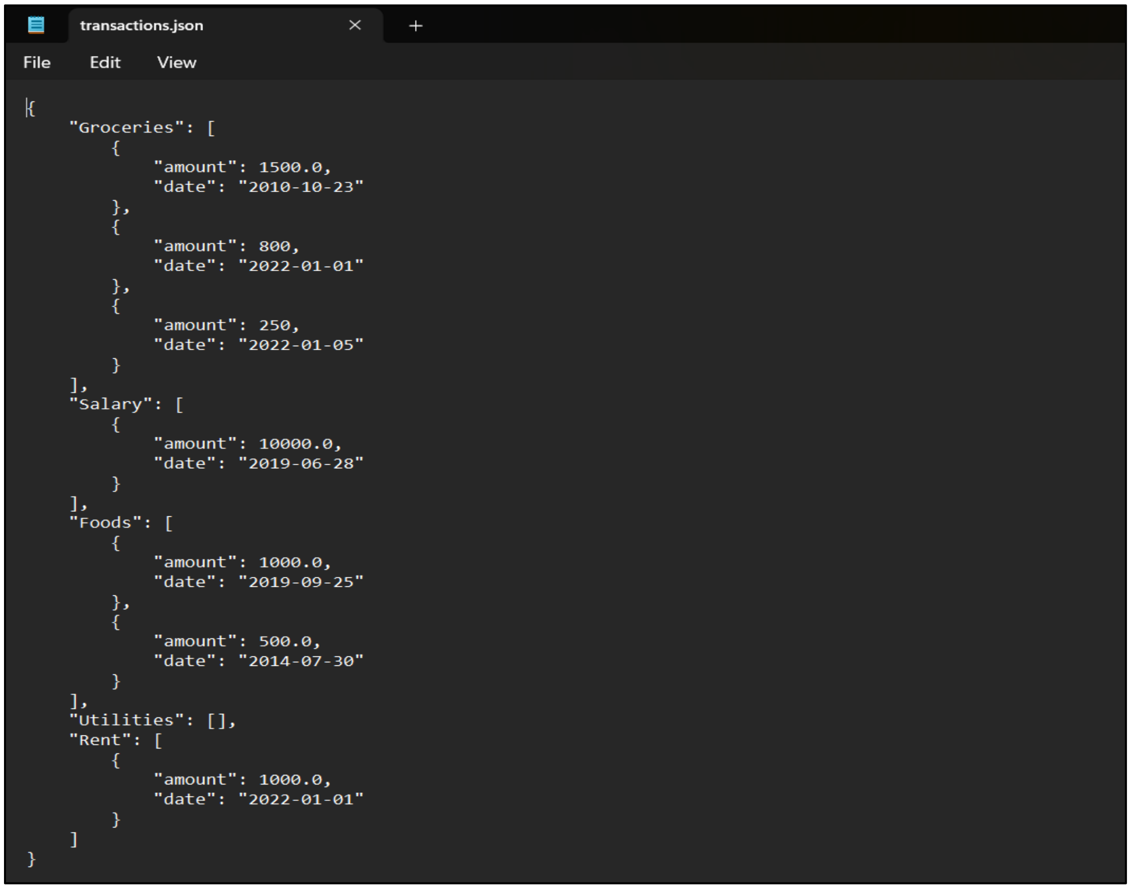
```
import tkinter as tk
from tkinter import ttk
import json
import os
class FinanceTrackerCLI:
    def __init__(self):
        self.filename = "transactions.json"
        self.transactions = self.load_transactions(self.filename)
    def load_transactions(self, filename):
        if os.path.exists(filename):
            with open(filename, "r")as file:
                return json.load(file)
        else:
            return {}

    def save_transactions(self, filename):
        with open(filename, "w")as file:
            json.dump(self.transactions, file, indent=4)

    def run_cli(self):
        while True:
            print("\nPersonal Finance Tracker")
            print("1. Show all transactions")
            print("2. Add a new transaction")
            print("3. Update a transaction")
            print("4. Delete transaction")
            print("5. Display Summary")
            print("6. Launch GUI")
            print("7. Exit")
            choice = input("Enter your choice: ")

            if choice == "1":
                self.show_transactions()
            elif choice == "2":
                self.add_transaction()
                self.save_transactions()
            elif choice == "3":
                self.update_transaction()
            elif choice == "4":
                self.delete_transaction()
                self.save_transactions()
            elif choice == "5":
                self.display_summary()
            elif choice == "6":
                self.launch_gui()
        elif choice == "7":
            self.save_transactions()
            print("Exiting the program")
            break
        else:
            print("Invalid choice.Please check again.")

def show_transactions(self):
    for category, transactions in self.transactions.item():
        print(f"\n{category}: ")
        for transaction in transactions:
            print(f"Amount: {transaction['amount']}, Date: {transaction['date']}")

def add_transaction(self):
    category = input("Enter the category: ")
    if cattegory not in self.transaction:
        self.transactions[category] = []
    date =input("Enter the transaction date(YYYY-MM-DD): ")
    amount = float(input("Enter the transaction amount: "))
    new_transaction = {"date": date, "amount": amount}
    self.transactions[category].append(new_transaction)
    self.save_transaction(self.filename)
    print("Transactions added successfully!")

def update_transaction(self):
    category = input("Enter the category of the transaction to update: ")
    if category in self.transactions:
        while True:
            try:
                index = int(input("Enter the index of the transaction to update: ")) - 1
                if 0 <= index < len(self.transactions[category]):
                    break
                else:
                    print("Invalid index.Please check again.")
            except ValueError:
                print("Invalid input.Enter a number.")

        while True:
            try:
                amount = float(input("Enter the new transaction amount: "))
                break
            except ValueError:
                print("Invalid input.Please check again.")

        date = input("Enter the transaction date(YYYY-MM-DD): ")
```

```
            self.transactions[category][index] = {"amount": amount, "date": date}
            print("Transaction updated successfully!")
        else:
            print("Category not found.")

    def delete_transaction(self):
        category = input("Enter the categor of the transaction to delete: ")
        if category in self.transactions:
            index = int(input("Enter the index of the  transaction to delete: ")) - 1
            if 0 <= index < len(self.transactions[category]):
                del self.transactions[category][index]
                print("Transaction deleted.")
            else:
                print("Invalid index.Try again.")
        else:
            print("Entered category not found.Please check again.")

    def display_summary(self):
        total_by_category = {}
        total_amount = 0

        for category,transactions_list in self.transactions.items():
            total_category = sum(transactions['amount'] for transactions in transactions_list)
            total_by_category[category] = total_category
            total_amount += total_category

        print("\nSummary")
        if not total_category:
            print("No transactions to display.")
        else:
            for category, amount in total_by_category.items():
                print(f"{category}: {amount}")

        print(f"Total: {total_amount}")
        print("End of summary")

    def launch_gui(self):
        root = tk.Tk()
        app = FinanceTrackerGUI(root,self.transactions)
        root.mainloop()
        self.transactions = app.transactions
        self.save_transactions(self.filename)

class FinanceTrackerGUI:
    def __init__(self, root,transactions):
        self.root = root
        self.root.title("Personal Finance Tracker")
        self.transactions = transactions
        self.sort_column = None
        self.sort_descending = False
        self.create_widgets()

    def create_widgets(self):
        self.table_frame = ttk.Frame(self.root)
        self.table_frame.pack(fill=tk.BOTH,expand=True)

        self.tree = ttk.Treeview(self.table_frame, columns=("Category", "Amount", "Date"),
show="headings")
        self.tree.heading("Category", text="Category", command= self.sort_by_category)
        self.tree.heading("Amount", text="Amount", command=self.sort_by_amount)
        self.tree.heading("Date", text="Date", command=self.sort_by_date)
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        self.scrollbar = ttk.Scrollbar(self.table_frame, orient="vertical", command=self.tree.yview)
        self.scrollbar.pack(side=tk.RIGHT, fill="y")

        self.tree.configure(yscrollcommand=self.scrollbar.set)


        self.search_var = tk.StringVar()

        self.search_entry = ttk.Entry(self.root, textvariable=self.search_var)

        self.search_button = ttk.Button(self.root, text="Search", command=self.search_transactions)

        self.search_entry.pack()

        self.search_button.pack()


        self.display_transactions(self.transactions)

    def load_transactions(self,filename):
        try:
            with open(filename, "r")as file:
                transactions =json.load(file)
            return transactions
        except FileNotFoundError:
            print("File not found.")
            return {}

    def display_transactions(self,transactions):
        #Remove existing entries
        for row in self.tree.get_children():
            self.tree.delete(row)

        #Add transactions to treeview
        for category, data in transactions.items():
            for transaction in data:
                self.tree.insert("", tk.END, values=(category,transaction["amount"],
transaction["date"]))

    def search_transactions(self):
        search_criteria = self.search_var.get()
        filtered_transactions = {}

        for category, data in self.transactions.items():
            filtered_data= [
                transaction
                for transaction in data
                if search_criteria.lower() in transaction["date"].lower()
                or str(search_criteria) in str(transaction["amount"])
                or search_criteria in category.lower()
        ]
            if filtered_data:
                filtered_transactions[category] = filtered_data

            self.display_transactions(filtered_transactions)

    def sort_by_column(self, col,reverse):
        if self.sort_column == col:
            self.sort_descending = not self.sort_descending
        else:
            self.sort_column = col
            self.sort_descending = False

        items = self.tree.get_children("")

        if col == "Amount":
            data =[(float(self.tree.set(item, col)), item) for item in items]
        else:
            data = [(self.tree.set(item, col), item) for item in items]
        data.sort(reverse=self.sort_descending)
        for index, (val, item) in enumerate(data):
            self.tree.move(item, "", index)
        self.sort_descending = reverse

    def sort_by_category(self):
        self.sort_by_column("Category", False)
    def sort_by_amount(self):
        self.sort_by_column("Amount",self.sort_descending)

    def sort_by_date(self):
        self.sort_by_column("Date", True)

    def save_transactions(self, filename):
        with open(filename, "w") as file:
            json.dump(self.transactions, file, indent=4)

def main():
    cli = FinanceTrackerCLI()
    cli.run_cli()

if __name__ == "__main__":
    main()
```

4. Test Cases.

4.1. Loading transactions.

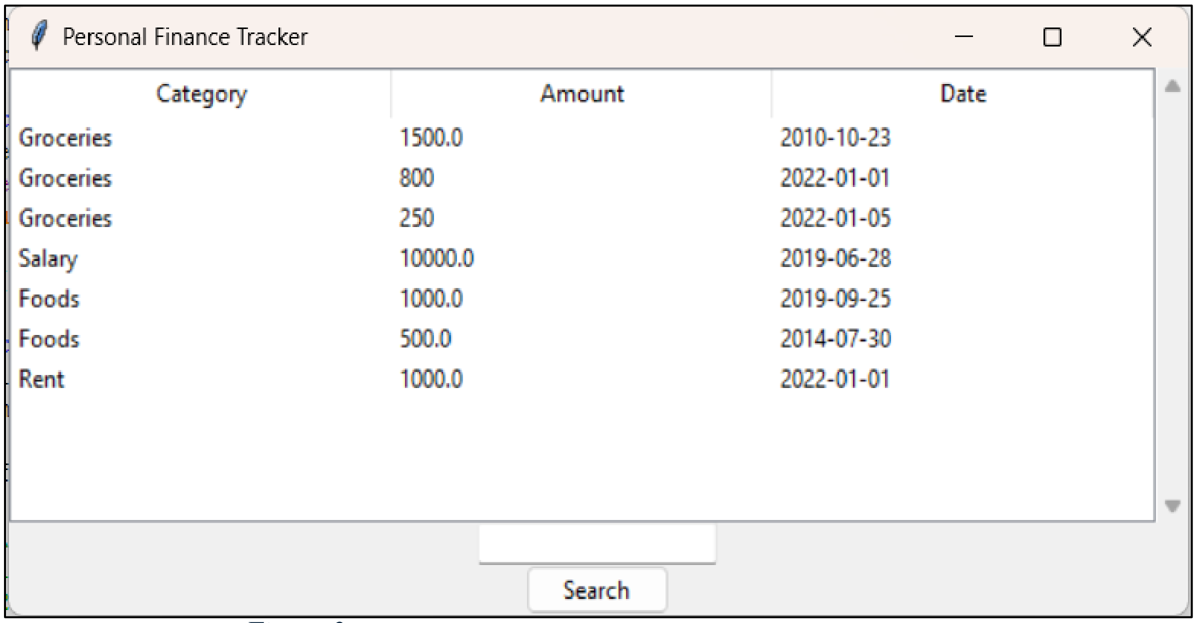
4.1.1. JSON file named as transactions.json



```
{
    "Groceries": [
        {
            "amount": 1500.0,
            "date": "2010-10-23"
        },
        {
            "amount": 800,
            "date": "2022-01-01"
        },
        {
            "amount": 250,
            "date": "2022-01-05"
        }
    ],
    "Salary": [
        {
            "amount": 10000.0,
            "date": "2019-06-28"
        }
    ],
    "Foods": [
        {
            "amount": 1000.0,
            "date": "2019-09-25"
        },
        {
            "amount": 500.0,
            "date": "2014-07-30"
        }
    ],
    "Utilities": [],
    "Rent": [
        {
            "amount": 1000.0,
            "date": "2022-01-01"
        }
    ]
}
```

Figure 1: JSON file.

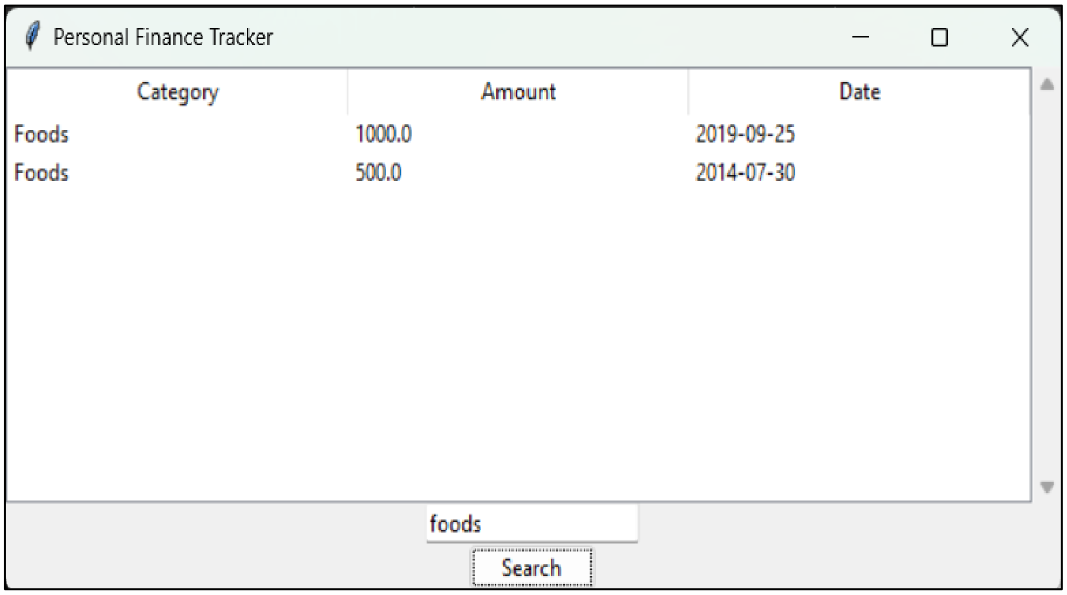
4.1.2. Output



Category	Amount	Date
Groceries	1500.0	2010-10-23
Groceries	800	2022-01-01
Groceries	250	2022-01-05
Salary	10000.0	2019-06-28
Foods	1000.0	2019-09-25
Foods	500.0	2014-07-30
Rent	1000.0	2022-01-01

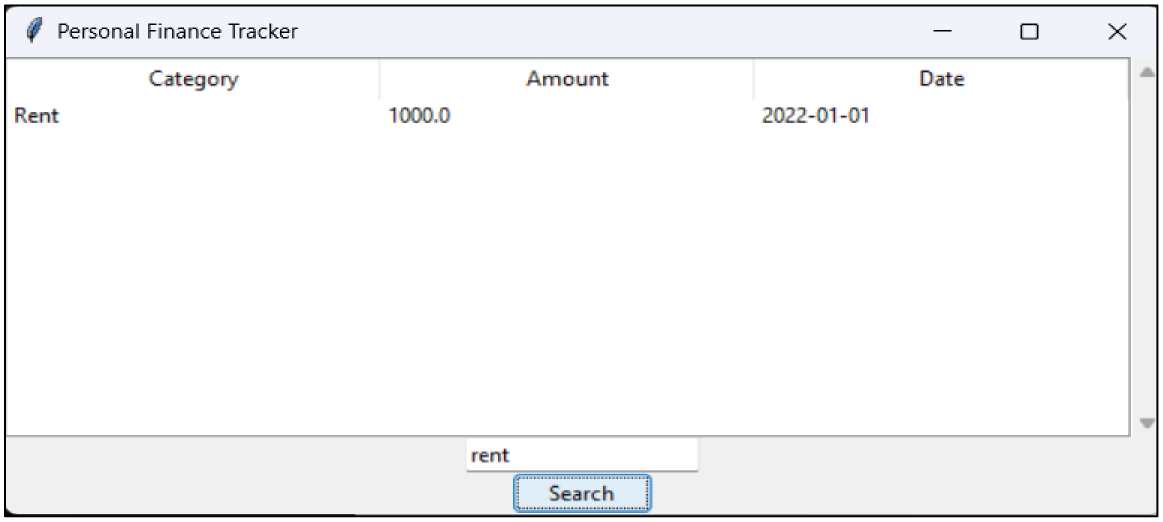
Figure 2:test case output.

4.2. Search Transactions in the table.



Category	Amount	Date
Foods	1000.0	2019-09-25
Foods	500.0	2014-07-30

Figure 3:test case 04.2.1.



Category	Amount	Date
Rent	1000.0	2022-01-01

Figure 4:test case 04.2.2.

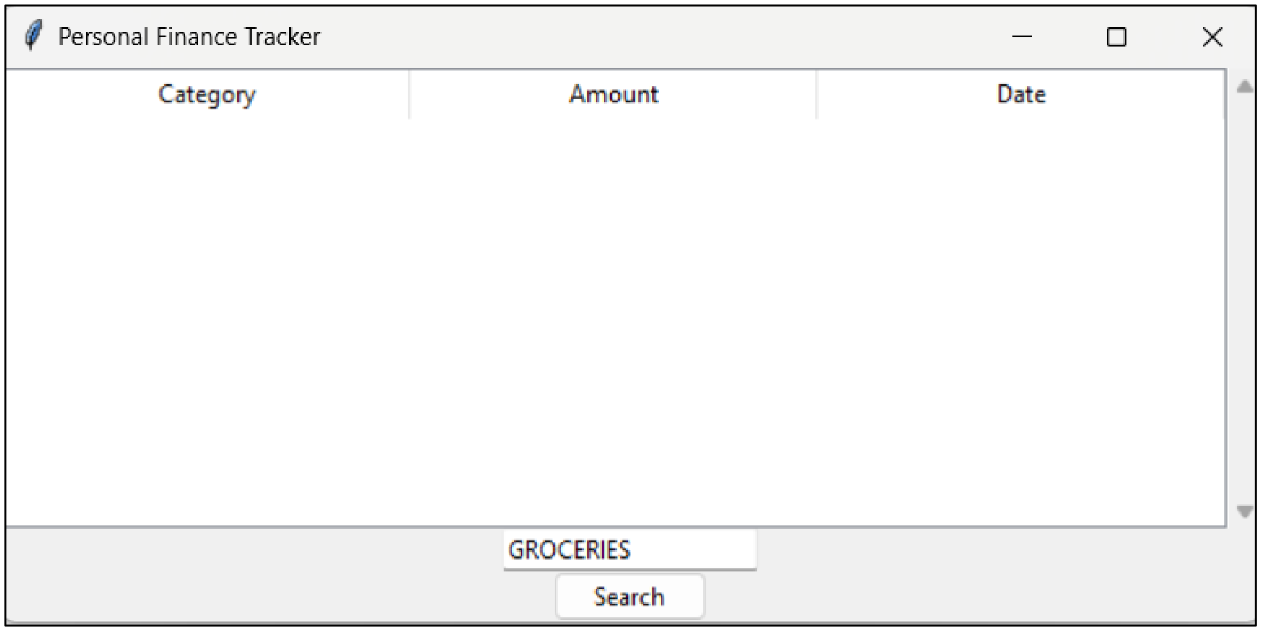
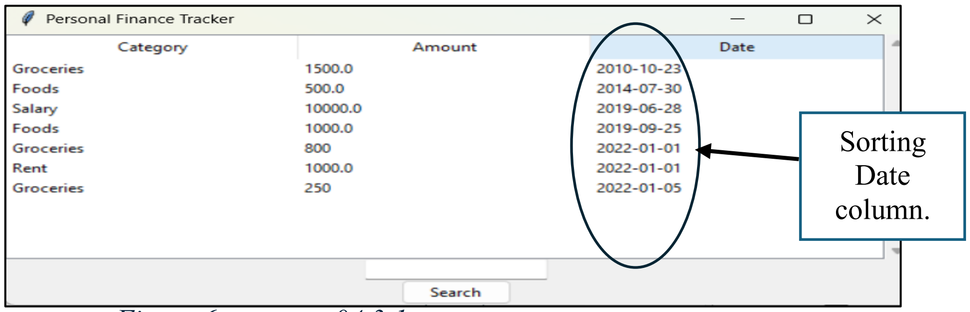


Figure 5:test case 04.2.3.

4.3. Sorting the transactions in the table.

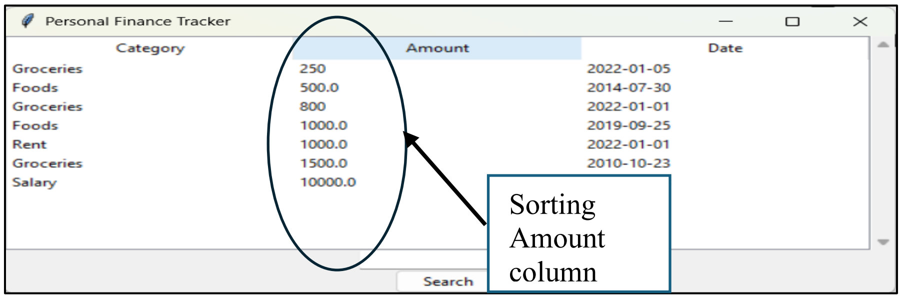
4.4.1. Sorting date column.



Category	Amount	Date
Groceries	1500.0	2010-10-23
Foods	500.0	2014-07-30
Salary	10000.0	2019-06-28
Foods	1000.0	2019-09-25
Groceries	800	2022-01-01
Rent	1000.0	2022-01-01
Groceries	250	2022-01-05

Figure 6:test case 04.3.1.

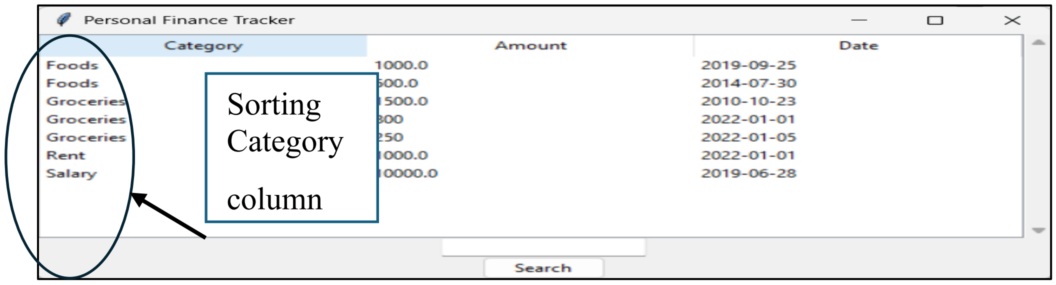
4.4.2. Sorting amount column.



Category	Amount	Date
Groceries	250	2022-01-05
Foods	500.0	2014-07-30
Groceries	800	2022-01-01
Foods	1000.0	2019-09-25
Rent	1000.0	2022-01-01
Groceries	1500.0	2010-10-23
Salary	10000.0	

Figure 7:test case 04.3.2.

4.4.3. Sorting category column.



Category	Amount	Date
Foods	1000.0	2019-09-25
Foods	500.0	2014-07-30
Groceries	1500.0	2010-10-23
Groceries	800	2022-01-01
Groceries	250	2022-01-05
Rent	1000.0	2022-01-01
Salary	10000.0	2019-06-28

Figure 8:test case 04.3.3.

5.How to use the program.

1. Make sure you have python installed on your system. If not download python using the official python website.
2. Tkinter should come pre-installed with python. If it’s not installed, you can install it using pip with command ‘pip install tk’
3. Copy the python code and save it into a file.
4. Open your terminal and run the code.
5. Once you run the code, the CLI for Personal Finance Tracker will be displayed in the terminal. You can interact with the CLI.
6. To launch the GUI, select option 6 from CLI menu. This will open a window with a graphical interface for managing transactions.
7. Follow the prompts in the GUI to add, update, delete transactions or display summary.

6.Design Decisions.

- Data storage: transactions are stored in a json file, which is easy to read format.
- Class structure:

  This code divided into two classes, ‘FinanceTrackerCLI’ and ‘FinanceTrackerGUI’, to separate the command line and graphical user interfaces logic.
- Error handling: the applications checks whether the transaction file exists before loading it, and it handles invalid user inputs.
- User interaction: CLI provides simple menu for user interaction, while GUI uses ‘tkinter’ for user interaction.

7. Class Structures.

FinanceTrackerCLI:

- ‘__init__’: initializes the class with the filename of the json file.
- ‘load_transactions’: loads transactions from the json file into memory.
- ‘save_transactions’: saves the current transactions to the json file.
- ‘run_cli’: runs the command-line interface, allowing user to interact.
- ‘show_transactions’, ‘add_transactions’, ‘update_transaction’, ‘delete_transaction’, ‘display_summary’: methods that allow the user to perform various operations.
- ‘launch_gui’: launches the graphical user interface.

FinanceTrackerGUI:

- ‘__init__’: initialize the GUI with the root’tkinter’ window and the transactions data.
- ‘create_widgets’: setup the GUI components, including treeview for displaying transactions, scrollbar, and search entry with a button.
- ‘load_transactions’: loads transactions from the json file.
- ‘display_transactions’: fills the treeview with transaction data.
- ‘search_transactions’: filters transactions based ouser input.
- ‘sort_by_column’: sort the transaction in the treeview based on the selected column.